# Declarative SQL vs. PL SQL

Database System Concepts, 5th Ed.



# Declarative SQL

- **Δηλωτικές ή μη διαδικαστικές (Declarative or nonprocedural)** – Ο χρήστης δηλώνει ποια δεδομένα απαιτούνται χωρίς να τον ενδιαφέρει ο τρόπος που θα ανακτηθούν αυτά τα δεδομένα (SQL)

```
select ....
from  ....
where P
```

P=Predicate (κατηγόρημα)
P= Συνδυασμός από λογικές και αριθμητικές εκφράσεις
P=Εμφωλιασμένο Υποερώτημα

? : exists/non exists
? : In/ Not In
? : Some, Any, All
? :Unique / Non Unique
……

Γενική δομή:

```
select  ...
from ...
where
      ? (select ...
                from ...
                where   ... );
```

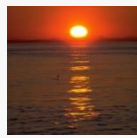
υποερώτηση

Insert into R
Select from
where P

Update R
where P

Delete R
where P

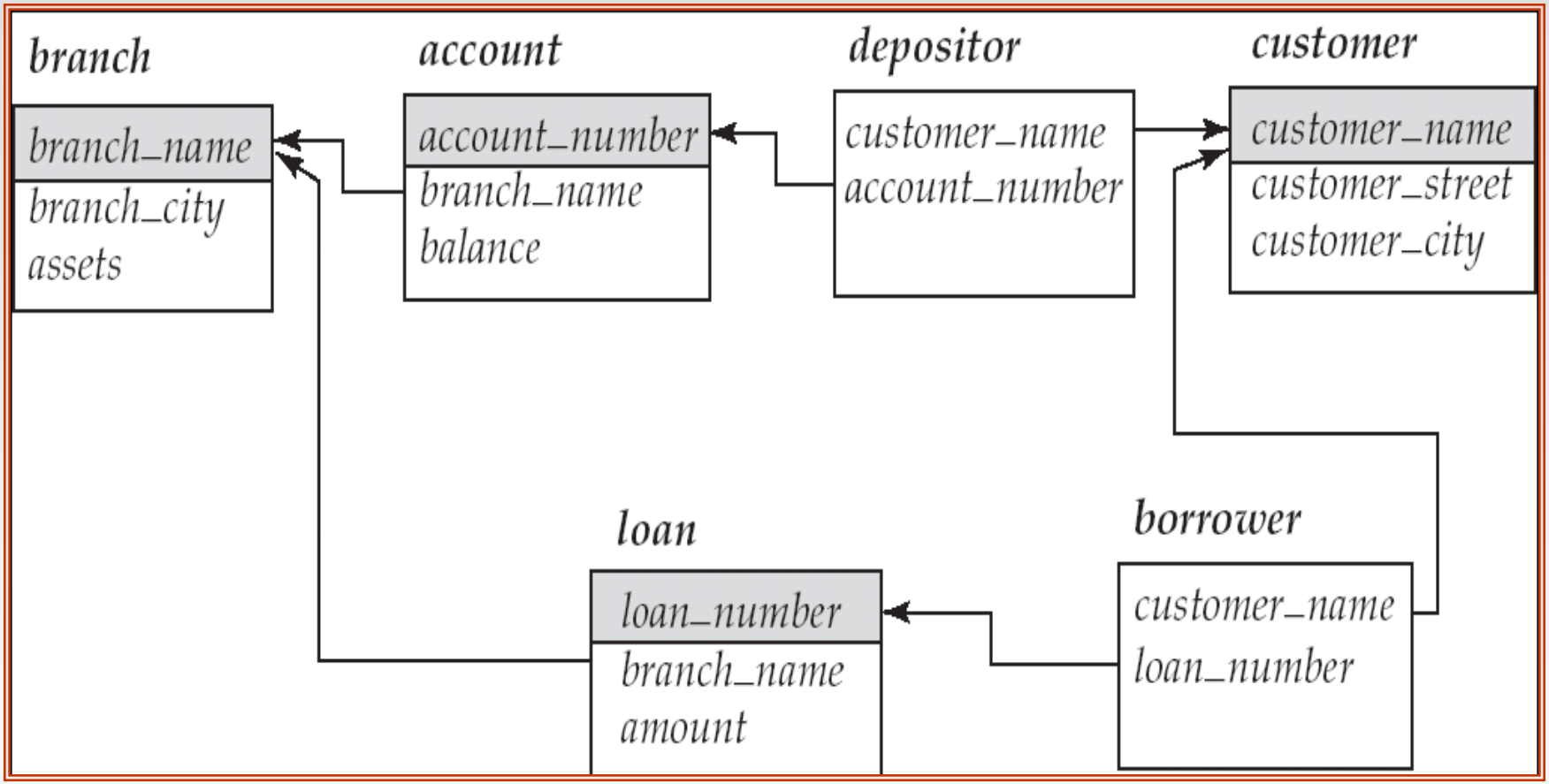
Υπολογισμός της υποερώτησης για κάθε γραμμή (πλειάδα) της εξωτερικής ερώτησης



# PL SQL

- Σκανδάλες (Triggers)
- Ενσωματωμένη (Embedded) SQL
- Δυναμική (Dynamic) SQL
- Συναρτήσεις και Διαδικαστικές Δομές (Functions and Procedural Constructs**)
- Αναδρομικά Ερωτήματα (Recursive Queries**)
- Προχωρημένα Θέματα SQL**

# PL SQL



# Trigger (Ειδοποίηση)

- "Υποθέστε ότι ο Jones κάνει αναλήψεις από κάποιο λογαριασμό που έκανε αρνητικό το υπόλοιπο του λογαριασμού του. Ας πούμε ότι το t δηλώνει την εγγραφή του λογαριασμού με αρνητική τιμή balance"
- Οι ενέργειες που θα πρέπει να κάνει αυτόματα το τραπεζικό σύστημα είναι οι εξής:
- (1) Να εισαχθεί μία νέα εγγραφή s στη σχέση loan με
  - s[loan_number]=t[account_number]
  - s[branch_name]=t[branch_name]
  - s[amount]= - t[balance]

  …..σιγά που δεν θα τον άφηνε να τραβήξει χρήματα…του τα χρεώνει τώρα σαν δάνειο……

  (2) Εισάγουμε μια νέα εγγραφή u στη σχέση borrower με
  - u[customer_name]="Jones"
  - u[loan_number]=t[account_number]

  (3) Ορίζουμε στον πίνακα account το t[balance]=0

# Trigger (Ειδοποίηση) (Συν.)

- **Create trigger** overdraft-trigger **after update on** account
  **referencing new row as** nrow
  **for each row**
  **When** nrow.balance < 0
  **begin atomic**
    **insert into** borrower
      (**select** customer_name, account_number
       **from** depositor
       **where** nrow.account_number=depositor.account-number);
    **insert into** loan **values**
      (nrow.acount_number, nrow.branch_name, - nrow.balance);
    **update** account **Set** balance=0
      **where** account.account_number=nrow.account-number)

***Κώδικας Trigger σε SQL:1999***

# Trigger (Ειδοποίηση) (Συν.)

- **Create trigger** overdraft-trigger **after update of** balance **on** account
  **referencing new row as** nrow
  **for each row**
  **When** nrow.balance < 0
  **begin atomic**
  **insert into** borrower
  (**select** customer_name, account_number
  **from** depositor
  **where** nrow.account_number=depositor.account-number);
  **insert into** loan **values**
  (nrow.acount_number, nrow.branch_name, - nrow.balance);
  **update** account **Set** balance=0
  **where** account.account_number=nrow.account-number)

***Κώδικας Trigger σε SQL:1999: Εδώ η ενεργοποίησή του γίνεται μετά από update σε συγκεκριμένη στήλη, για την ακρίβεια τη στήλη balance***

# Trigger (Ειδοποίηση) (Συν.)

- **Create trigger** overdraft-trigger **on** account
  **for update**
  **as**
  **if inserted**.balance < 0
  **begin**
  **insert into** borrower
  (**select** customer_name, account_number
  **from** depositor, **inserted**
  **where inserted**.account_number=depositor.account-number);
  **insert into** loan **values**
  **(inserted**.acount_number, **inserted**.branch_name, - **inserted**. balance);
  **update** account **Set** balance=0
  **where** account.account_number=**inserted**.account-number)

***Κώδικας Trigger σε MS–SQL Server***

# Ενσωματωμένη (Embedded) SQL

- Το SQL πρότυπο (standard) ορίζει ενσωματώσεις της γλώσσας SQL σε μία πλειάδα γλωσσών προγραμματισμού όπως C, Java, και Cobol.
- Η γλώσσα στην οποία τα SQL queries ενσωματώνονται είναι γνωστή με το όνομα **host language**, και οι SQL δομές που επιτρέπονται μέσα σε μία host language αποτελούν την επονομαζόμενη *embedded* SQL.
- Η βασική μορφή αυτών των γλωσσών ακολουθεί το System R το οποίο ενσωματώνει SQL μέσα σε PL/I.
- Η **EXEC SQL** εντολή χρησιμοποιείται για να δηλώσει στον προ-επεξεργαστή (preprocessor) αίτημα ενσωματωμένης SQL

  EXEC SQL <embedded SQL statement > END_EXEC

  Σημείωση: Αυτό διαφοροποιείται από γλώσσα σε γλώσσα (για παράδειγμα, στη γλώσσα Java η ενσωμάτωση (embedding) γίνεται ως εξής:

  # SQL { …. }; )

# Example Query

- *Μέσα από μία host language, βρες τα ονόματα και τις πόλεις των πελατών που έχουν σε κάποιο λογαριασμό υπόλοιπο μεγαλύτερο από την τιμή της μεταβλητής amount*.

- Προδιαγράφουμε το query σε SQL και δηλώνουμε έναν *cursor* γι' αυτό

```
EXEC SQL
  declare c cursor for
  select customer_name, customer_city
  from depositor, customer, account
  where depositor.customer_name = customer.customer_name
      and depositor account_number = account.account_number
       and account.balance > :amount
END_EXEC
```

# Embedded SQL (Συν.)

- Η εντολή **open** προκαλεί την έναρξη εκτέλεσης του ερωτήματος

  EXEC SQL **open** *c* END_EXEC

- Η εντολή **fetch** τοποθετεί τις τιμές ενός tuple του αποτελέσματος σε μεταβλητές της οικοδέσποινας γλώσσας (host language).

  EXEC SQL **fetch** *c* **into** :*cn, :cc* END_EXEC

  Επαναλαμβανόμενες κλήσεις της **fetch** τοποθετούν ένα-ένα τα tuples του αποτελέσματος

- Μία δεσμευμένη μεταβλητή SQLSTATE (της κλάσης SQLCA) παίρνει την τιμή '02000' για να υποδηλώσει ότι δεν υπάρχουν διαθέσιμα άλλα δεδομένα για να γίνουν fetch, δηλαδή να φορτωθούν στις μεταβλητές *cn* και *cc.*

- Η εντολή **close** προκαλεί στο database σύστημα να διαγράψει την προσωρινή σχέση η οποία έχει αποθηκεύσει το αποτέλεσμα του ερωτήματος.

  EXEC SQL **close** *c* END_EXEC

Σημ: Οι παραπάνω λεπτομέρειες ποικίλουν από γλώσσα σε γλώσσα. Π.χ., στην Java δεν χρειάζονται επαναλαμβανόμενες κλήσεις της fetch, αφού η ίδια η γλώσσα ορίζει αυτόματα ειδικούς Java iterators που κάνουν την παραπάνω δουλειά.

# Ενημερώσεις δια μέσω Cursors

- Μπορούμε να ενημερώσουμε το tuple που γίνεται fetched από τον cursor δηλώνοντας ότι ο cursor αυτός είναι για update:

```
declare c cursor for
   select *
   from account
   where branch_name = 'Perryridge'
for update
```

- Για να ενημερώσουμε το tuple στην τρέχουσα θέση του cursor *c*, *γράφουμε:*

```
update account
set balance = balance + 100
where current of c
```

# Δυναμική SQL

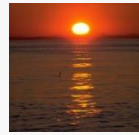
- Επιτρέπει σε προγράμματα να δομούν και να αποστέλλουν SQL queries κατά το χρόνο εκτέλεσής τους (at run time).
- Παράδειγμα χρήσης δυναμικής SQL μέσα από πρόγραμμα C.

```
char *  sqlprog = "update account
                   set balance = balance * 1.05
                    where account_number = ?"
EXEC SQL prepare dynprog  from :sqlprog;
char account [10] = "A-101";
EXEC SQL execute dynprog using :account;
```

- Ένα δυναμικό SQL πρόγραμμα περιέχει ένα ?, ο οποίος είναι ένας place holder για μία τιμή (value) η οποία μας *προμηθεύεται όταν ένα* SQL πρόγραμμα εκτελείται.
- *** Dynamic SQL is a programming technique that enables you to build SQL statements dynamically at runtime. You can create more general purpose, flexible applications by using dynamic SQL because **the full text of a SQL statement may be unknown at compilation**. **For example, dynamic SQL lets you create a procedure that operates on a table whose name is not known until runtime.**

# ODBC και JDBC

- API (application-program interface): Ένα πρόγραμμα που θέλει να αλληλεπιδράσει με έναν database server
- Η εφαρμογή (Application) κάνει κλήσεις για να
  - Συνδεθεί με τον database server
  - Αποστείλει SQL εντολές στον database server
  - Κάνει Fetch τα tuples του αποτελέσματος ένα-ένα (one-by-one) σε μεταβλητές του προγράμματος
- ODBC (Open Database Connectivity) συνεργάζεται με C, C++, C#, και Visual Basic
- JDBC (Java Database Connectivity) συνεργάζεται αποκλειστικά με Java



# ODBC

- Open DataBase Connectivity(ODBC) standard
  - Πρότυπο ενός *προγράμματος – εφαρμογής* προκειμένου το τελευταίο να συνδεθεί με έναν database server.
  - Το API (Application Program Interface) κάνει τα εξής:
    - Κάνει open ένα connection με την database,
    - Αποστέλλει queries και updates,
    - Παίρνει πίσω τα αποτελέσματα.
- Εφαρμογές όπως GUI, spreadsheets, κ.τ.λ. μπορούν να χρησιμοποιήσουν ODBC

# ODBC (Συν.)

- Κάθε σύστημα ΒΔ που υποστηρίζει ODBC παρέχει στην ουσία μία "driver" βιβλιοθήκη (library) η οποία γίνεται linked με το client program.
- Όταν το client program κάνει μία ODBC API κλήση, ο κώδικας της βιβλιοθήκης επικοινωνεί με τον server για να εκτελέσει την απαιτούμενη ενέργεια, και να φορτώσει (fetch) τα αποτελέσματα.
- Το ODBC πρόγραμμα πρώτα δεσμεύει ένα SQL περιβάλλον, εν συνεχεία μία λαβή (handle) επικοινωνίας με τη ΒΔ.
- Ανοίγει την επικοινωνία με τη ΒΔ εκτελώντας την εντολή SQLConnect(). Οι παράμετροι της SQLConnect είναι:
  - Χερούλι επικοινωνίας (connection handle),
  - Ο server με τον οποίο θα γίνει η σύνδεση
  - Η ταυτότητα του χρήστη (user identifier)
  - Ο κωδικός (password)
- Πρέπει επίσης να οριστούν οι τύποι των ορισμάτων:
  - SQL_NTS δηλώνει ότι το προηγούμενο όρισμα είναι ένα null-terminated string.

# ODBC Κώδικας

- int ODBCexample()

```
{
    RETCODE error;
    HENV    env;     /* environment */
    HDBC    conn;  /* database connection */
    SQLAllocEnv(&env);
    SQLAllocConnect(env, &conn);
    SQLConnect(conn, "aura.bell-labs.com", SQL_NTS, "avi", SQL_NTS,
      "avipasswd", SQL_NTS);
    { …. Do actual work … }

    SQLDisconnect(conn);
    SQLFreeConnect(conn);
    SQLFreeEnv(env);
}
```

# ODBC Κώδικας (Συν.)

- Το πρόγραμμα στέλνει SQL queries στη ΒΔ χρησιμοποιώντας το *SQLExecDirect*
- Τα tuples του αποτελέσματος φορτώνονται με τη χρήση της SQLFetch()
- Η *SQLBindCol()* κάνει bind σε C μεταβλητές τα πεδία (attributes) του query result
  - Όταν ένα tuple γίνεται fetched, οι τιμές των πεδίων του αυτόματα φορτώνονται στις κατάλληλες C μεταβλητές.
  - Τα ορίσματα της *SQLBindCol()* είναι:
    - ODBC stmt μεταβλητή, η θέση του πεδίου στο query result
    - Ο τύπος *conversion from SQL to C*.
    - Η διεύθυνση της μεταβλητής.
    - Για τύπους μεταβλητών, όπως πίνακες χαρακτήρων,
      - Το μέγιστο μήκος της μεταβλητής
      - Η θέση στην οποία θα αποθηκευτεί το ακριβές μήκος όταν ένα tuple γίνεται fetched.
      - Σημ: Αρνητική τιμή επιστρέφεται για μήκος πεδίου με τιμή null

# ODBC Code (Cont.)

- Κύριο Σώμα ενός προγράμματος

```
char branchname[80];
float  balance;
int  lenOut1, lenOut2;
HSTMT   stmt;

SQLAllocStmt(conn, &stmt);
char * sqlquery = "select branch_name, sum (balance)
                   from account
                   group by branch_name";

error = SQLExecDirect(stmt, sqlquery, SQL_NTS);

if (error == SQL_SUCCESS)
{
    while (SQLFetch(stmt) >= SQL_SUCCESS)
     {
SQLBindCol(stmt, 1, SQL_C_CHAR,   branchname , 80, &lenOut1);
        SQLBindCol(stmt, 2, SQL_C_FLOAT, &balance, 0,&lenOut2);
            printf (" %s  %g\n", branchname, balance);
        }
}
    SQLFreeStmt(stmt, SQL_DROP);
```

# Περισσότερα ODBC Features

- Εξ' ορισμού, το σύστημα ΒΔ μεταχειρίζεται κάθε SQL εντολή ως μία ξεχωριστή συναλλαγή (transaction) η οποία διεκπεραιώνεται (committed).
  - Μπορεί να κλείσει αυτόματα την διεκπεραίωση σε μία σύνδεση
    - SQLSetConnectOption(conn, SQL_AUTOCOMMIT, 0)}
  - Οι συναλλαγές τότε πρέπει να γίνουν ρητά είτε *committed* ή *rolled back*
    - SQLTransact(conn, SQL_COMMIT) or
    - SQLTransact(conn, SQL_ROLLBACK)

# JDBC

- Το **JDBC** είναι ένα Java API για επικοινωνία με database systems τα οποία υποστηρίζουν SQL
- Το JDBC υποστηρίζει μία πλειάδα από τρόπους βάσει των οποίων μπορούμε να ρωτήσουμε και να ενημερώσουμε δεδομένα, καθώς και να ανακτήσουμε query results
- Το JDBC υποστηρίζει επίσης ανάκτηση μεταδεδομένων (metadata), όπως ερωτήματα σχετικά με τις υπάρχουσες σχέσεις (relations) σε μία ΒΔ, καθώς επίσης τα ονόματα και τους τύπους των πεδίων των σχέσεων (relation attributes)
- Μοντέλο επικοινωνίας με τη ΒΔ:
  - Κάνει Open ένα connection
  - Δημιουργεί ένα “statement” object
  - Εκτελεί queries χρησιμοποιώντας το Statement object για να αποστείλει queries και να λάβει (fetch) τα αποτελέσματα
  - Μηχανισμοί Εξαιρέσεων για χειρισμό λαθών

# JDBC Κώδικας

```
public static void JDBCexample(String dbid, String userid, String passwd)
{
    try {
        Class.forName ("oracle.jdbc.driver.OracleDriver");
        Connection conn = DriverManager.getConnection(
          "jdbc:oracle:thin:@aura.bell-labs.com:2000:bankdb", userid, passwd);
        Statement stmt = conn.createStatement();
          … Do Actual Work ….
        stmt.close();
        conn.close();
    }
    catch (SQLException sqle) {
        System.out.println("SQLException : " + sqle);
    }
}
```

# JDBC Code (Cont.)

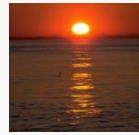
- Ενημέρωσε μία ΒΔ

```
try {
    stmt.executeUpdate(  "insert into account values
                                ('A-9732', 'Perryridge', 1200)");
} catch (SQLException sqle) {
    System.out.println("Could not insert tuple. " + sqle);
}
```

- Εκτέλεσε ερώτημα, λάβε και εκτύπωσε τα αποτελέσματα

```
ResultSet rset = stmt.executeQuery( "select branch_name,
    avg(balance)
                                from account
                                group by branch_name");
while (rset.next()) {
     System.out.println(
              rset.getString("branch_name") + "  " + rset.getFloat(2));

}
```



# Λεπτομέρειες JDBC Κώδικα

- Λαμβάνοντας result fields:
  - **rs.getString(“branchname”) and rs.getString(1) equivalent if branchname is the first argument of select result.**
- Πώς χειριζόμαστε τις Null τιμές

  **int a = rs.getInt(“a”);**

  **if (rs.wasNull()) Systems.out.println(“Got null value”);**

# Διαδικαστικές Επεκτάσεις & Stored Procedures

- Η SQL παρέχει μία **module** (**transact SQL**) γλώσσα η οποία:
  - Επιτρέπει τον ορισμό procedures στην SQL, με if-then-else εντολές, for και while επαναλήψεις (loops), κ.τ.λ.
  - Περισσότερα στο Κεφάλαιο 9
- Οι Stored Procedures
  - Μπορούν να αποθηκεύσουν *διαδικασίες* μέσα σε μία ΒΔ
  - Και εν συνεχεία να τις εκτελέσουν χρησιμοποιώντας μία **call** εντολή
  - Επιτρέπουν επίσης σε εξωτερικές εφαρμογές να συνεργάζονται με τη ΒΔ χωρίς να χρειάζεται να γνωρίζουν εσωτερικές λεπτομέρειες της ΒΔ.
- Όλα τα παραπάνω καλύπτονται εκτενώς στο Κεφάλαιο 9 (***Object Relational Databases***)

# Functions και Procedures

- Η SQL:1999 υποστηρίζει functions και procedures
  - Οι Functions/procedures μπορεί να αναπτυχθούν είτε σε SQL ή σε μία εξωτερική γλώσσα προγραμματισμού
  - Οι Functions είναι ιδιαίτερα χρήσιμες με ειδικούς τύπους δεδομένων όπως εικόνες και γεωμετρικά αντικείμενα
    - Παράδειγμα: functions που ελέγχουν αν επικαλύπτονται πολύγωνα ή που συγκρίνουν εικόνες για να ελέγξουν ομοιότητα (similarity)
  - Κάποια Συστήματα ΒΔ υποστηρίζουν **table-valued functions**, οι οποίες επιστρέφουν μία σχέση (relation) ως αποτέλεσμα
- Η SQL:1999 υποστηρίζει επίσης ένα πλούσιο σύνολο από αναγκαίες δομές, όπως:
  - Loops, if-then-else, assignment
- Πολλές ΒΔ έχουν ιδιόκτητες procedural extensions σε SQL οι οποίες διαφέρουν από την SQL:1999

# SQL Functions

- Όρισε μία συνάρτηση η οποία να παίρνει ως παράμετρο το όνομα ενός πελάτη, και να επιστρέφει το πλήθος των λογαριασμών που κατέχει ο συγκεκριμένος πελάτης.

```
create function account_count (customer  varchar(20))
returns integer
begin
    declare a_count integer;
    select count (* ) into a_count
    from depositor
    where depositor.customer_name = customer
    return a_count;
 end
```

# Table Functions

- Η SQL:2003 περιέχει συναρτήσεις οι οποίες επιστρέφουν ως αποτέλεσμα μία ολόκληρη σχέση (relation)
- Παράδειγμα: Επέστρεψε όλους τους λογαριασμούς που ανήκουν σε έναν συγκεκριμένο πελάτη.

```
create function accounts_of (customer  char(20)
        returns table (     account_number char(10),
                            branch_name char(15),
                            balance numeric(12,2))
return table
        (select account_number, branch_name, balance
         from account
         where exists (
          select *
          from depositor
          where depositor.customer_name = customer
          and depositor.account_number = account.account_number ))
```

# Table Functions (Συν.)

- Βρες τα ονόματα και τις διευθύνσεις κάθε πελάτη ο οποίος έχει περισσότερους από έναν λογαριασμούς.

**create function** *names_streets* (*customer* **varchar**(20)

**returns table** ( *customer_name* **char**(20),

*customer_street* **char**(20),

*customer_city* **char**(30))

**return table**

**(select** *customer_name, customer_street, customer_city*
**from** *customer*
**where** *account*_count (*customer* ) > 1)

# Table Functions (Συν.)

- Τρόπος χρήσης

  **select ***

  **from table** (*accounts_of* (‘Smith’))

# SQL Procedures

- Η συνάρτηση *account_count* μπορούσε να γραφτεί και ως διαδικασία (procedure) ως εξής:

```
create procedure account_count_proc (in customer  varchar(20),
                                     out a_count integer)
begin
  select count(*) into a_count
  from depositor
  where depositor.customer_name = customer
end
```

- Οι Procedures μπορούν να *κληθούν* είτε από μία SQL procedure ή από μία embedded SQL, χρησιμοποιώντας την εντολή **call**.

```
declare a_count integer;
call account_count_proc( 'Smith', a_count);
```

  Procedures και functions μπορούν να *κληθούν* επίσης από dynamic SQL

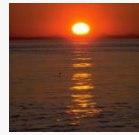
- Η SQL:1999 επιτρέπει σε περισσότερες από μία function/procedure να έχουν το ίδιο όνομα (**overloading**), εφόσον ο αριθμός των ορισμάτων (arguments) διαφέρει, ή τουλάχιστον διαφέρουν οι τύποι των ορισμάτων τους.

# Διαδικαστικές Δομές

- Η ενιαία δήλωση: **begin ... end**,
  - Μπορεί να περιέχει πολλαπλά SQL statements μεταξύ **begin** και **end**.
  - Καθώς και τοπικές μεταβλητές (Local variables)
- **While** και **repeat** statements:

```
declare n integer default 0;
while n < 10 do
    set n = n + 1
end while

repeat
    set n = n − 1
until n = 0
end repeat
```



# Διαδικαστικές Δομές (Συν.)

- **For** loop
  - Επιτρέπει επαναλήψεις πάνω στα αποτελέσματα ενός query
  - Παράδειγμα: Βρες το σύνολο των υπολοίπων στο υποκατάστημα Perryridge

```
declare n  integer default 0;
for r  as
      select balance from account
       where branch_name = 'Perryridge'
 do
      set n = n + r.balance
 end for
```

# Διαδικαστικές Δομές (Συν.)

- Εντολές συνθήκης (**if-then-else**)
  Π.χ. Για να κατατάξουμε το συνολικό υπόλοιπο κάθε πελάτη σε ποια από τις παρακάτω κατηγορίες ανήκει (balance <1000, >=1000 and <5000, >= 5000)

  **if** *r.balance* < 1000
  **then set** *l = l + r.balance*
  **elseif** *r.balance* < 5000
  **then set** *m = m + r.balance*
  **else set** *h = h + r.balance*
  **end if**

- Η SQL:1999 υποστηρίζει επίσης την εντολή **case** με παρόμοιο τρόπο όπως στη C
- Επισήμανση συνθηκών εξαίρεσης (exception conditions), και δηλώσεις χειρισμού εξαιρέσεων (exception handlers)

  **declare** *out_of_stock* **condition**
  **declare exit handler for** *out_of_stock*
  **begin**
  …
  .. **signal** out-of-stock
  **end**

  - Ο handler εδώ είναι το **exit** – προκαλεί έξοδο από το **begin..end** σώμα

# Εξωτερικές Functions/Procedures

- Η SQL:1999 επιτρέπει τη χρήση συναρτήσεων / διαδικασιών οι οποίες έχουν γραφτεί σε άλλη γλώσσα όπως C ή C++
- Πώς δηλώνουμε external language procedures και functions

```
create procedure account_count_proc(in customer  varchar(20),
                                    out count integer)
language C
external name ' /usr/avi/bin/account_count_proc'

create function account_count(customer  varchar(20))
returns integer
language C
external name '/usr/avi/bin/account_count'
```

# Αναδρομή (Recursion) στην SQL

- SQL:1999 επιτρέπει τον ορισμό των recursive views
- Παράδειγμα: Βρες όλα τα employee-manager ζευγάρια, όπου ο employee δίνει αναφορά στον manager άμεσα ή έμμεσα (δηλαδή ο manager του manager, ο manager του manager του manager, κ.τ.λ.)

```
with recursive empl (employee_name, manager_name ) as (
        select employee_name, manager_name
        from   manager
    union
        select manager.employee_name, empl.manager_name
        from   manager, empl
        where manager.manager_name = empl.employe_name)
select *
from   empl
```

# Example of Fixed-Point Computation

| *employee_name* | *manager_name* |
|---|---|
| Alon | Barinsky |
| Barinsky | Estovar |
| Corbin | Duarte |
| Duarte | Jones |
| Estovar | Jones |
| Jones | Klinger |
| Rensal | Klinger |

| *Iteration number* | *Tuples in empl* |
|---|---|
| 0 | |
| 1 | (Duarte), (Estovar) |
| 2 | (Duarte), (Estovar), (Barinsky), (Corbin) |
| 3 | (Duarte), (Estovar), (Barinsky), (Corbin), (Alon) |
| 4 | (Duarte), (Estovar), (Barinsky), (Corbin), (Alon) |

# SQL Server – Simple Recursive Query Example

In this post I use the common example of a table with **countries, states, and cities** and where we want to get **a list of all cities** in a **single country**. Enjoy!

Create a table called “Area”:

```
1  CREATE TABLE dbo.Area(
2    AreaID int NOT NULL,
3    AreaName varchar(100) NOT NULL,
4    ParentAreaID int NULL,
5    AreaType varchar(20) NOT NULL
6  CONSTRAINT PK_Area PRIMARY KEY CLUSTERED
7  ( AreaID ASC
8  ) ON [PRIMARY])
9  GO
```

```
INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(1, 'Canada', null, 'Country')

INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(2, 'United States', null, 'Country')

INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(3, 'Saskatchewan', 1, 'State')

INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(4, 'Saskatoon', 3, 'City')

INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(5, 'Florida', 2, 'State')

INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(6, 'Miami', 5, 'City')
```

# SQL Server – Simple Recursive Query Example

If I do a select by AreaType “City”:
I get both Saskatoon and Miami:

```
1 select * from dbo.Area
2 where AreaType = 'City'
```

| | AreaID | AreaName | ParentAreaID | AreaType |
|---|---|---|---|---|
| 2 | 4 | Saskatoon | 3 | City |
| 3 | 6 | Miami | 5 | City |

**However, what if I wanted to return all cities in Canada?**
You can accomplish this by doing a recursive select which uses a common table expression (CTE).

```
INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(1, 'Canada', null, 'Country')

INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(2, 'United States', null, 'Country')

INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(3, 'Saskatchewan', 1, 'State')

INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(4, 'Saskatoon', 3, 'City')

INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(5, 'Florida', 2, 'State')

INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(6, 'Miami', 5, 'City')
```

# SQL Server – Simple Recursive Query Example

```
WITH AreasCTE AS
(
--anchor select, start with the country of Canada, which will be the root element for our search
SELECT AreaID, AreaName, ParentAreaID, AreaType
FROM dbo.Area
WHERE AreaName = 'Canada'
UNION ALL
--recursive select, recursive until you reach a leaf (an Area which is not a parent of any other area)
SELECT a.AreaID, a.AreaName, a.ParentAreaID, a.AreaType
FROM dbo.Area a
INNER JOIN AreasCTE s ON a.ParentAreaID = s.AreaID
)
--Now, you will have all Areas in Canada, so now let's filter by the AreaType "City"
SELECT * FROM AreasCTE
where AreaType = 'City'
```

Now we get back the following results for cities in Canada:

| 1 | AreaID | AreaName | ParentAreaID | AreaType |
|---|---|---|---|---|
| 2 | 4 | Saskatoon | 3 | City |

```
INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(1, 'Canada', null, 'Country')

INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(2, 'United States', null, 'Country')

INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(3, 'Saskatchewan', 1, 'State')

INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(4, 'Saskatoon', 3, 'City')

INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(5, 'Florida', 2, 'State')

INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(6, 'Miami', 5, 'City')
```

# Advanced PL SQL Features**

- Δημιούργησε έναν πίνακα με το ίδιο σχήμα ενός υπάρχοντα:

  **create table** *temp_account* **like** *account*

- SQL:2003 επιτρέπει subqueries να λαβαίνουν χώρα οπουδήποτε μία τιμή απαιτείται, δεδομένου ότι το subquery επιστρέφει μία τιμή μόνο. Αυτό εφαρμόζεται εξίσου καλά στα updates.
- SQL:2003 επιτρέπει subqueries στην πρόταση **from** να προσπελαύνουν attributes από άλλες relations χρησιμοποιώντας τη δομή **lateral** (πλευρική παράλληλη):

  **select** *customer_name, num_accounts*

  **from** *customer*, **lateral** (

  **select count**(*)

  **from** *account*

  **where** *account.customer_name = customer.customer_name* )

  **as** *this_customer* (*num_accounts* )

# Advanced SQL Features (Συν.)

- Η δομή Merge επιτρέπει μαζική επεξεργασία (batch processing) ενημερώσεων.
- Παράδειγμα: Η σχέση ***funds_received* (*account_number, amount* )** έχει μία ομάδα / παρτίδα (batch) από καταθέσεις που πρέπει να προστεθούν στον κατάλληλο λογαριασμό (account) της σχέσης *account*

  **merge into** *account* **as** *A*

  **using** (**select** *

  **from** *funds_received* **as** *F* )

  **on** (A.*account_number* = *F.account_number* )

  **when matched then**

  **update set** *balance* = *balance* + *F.amount*

# ΤΕΛΟΣ ΚΕΦΑΛΑΙΟΥ

**Database System Concepts, 5th Ed.**